

# ASSIGNMENT-3.3

2303A51337

BATCH-10

## ➤ TASK-1:

### **PROMPT:**

PYTHON code for reading consumer details and generate a python program it should read previous units and current units and type of customer (domestic or commercial) and calculate the units consumed program without function

Reading consumer details and calculating units consumed

### **CODE:**

```
previous_units = int(input("Enter previous meter reading: "))

current_units = int(input("Enter current meter reading: "))

customer_type = input("Enter customer type (domestic/commercial): ").strip().lower()

units_consumed = current_units - previous_units

if units_consumed < 0:

    print("Error: Current units cannot be less than previous units.")

else:

    if customer_type == "domestic":

        if units_consumed <= 100:

            bill_amount = units_consumed * 1.5

        elif units_consumed <= 300:

            bill_amount = (100 * 1.5) + (units_consumed - 100) * 2.5

        else:

            bill_amount = (100 * 1.5) + (200 * 2.5) + (units_consumed - 300) * 4.0

    elif customer_type == "commercial":

        if units_consumed <= 100:

            bill_amount = units_consumed * 2.0

        elif units_consumed <= 300:
```

```
bill_amount = (100 * 2.0) + (units_consumed - 100) * 3.5

else:

    bill_amount = (100 * 2.0) + (200 * 3.5) + (units_consumed - 300) * 5.0

else:

    print("Error: Invalid customer type.")

    bill_amount = None
```

```
if bill_amount is not None:
```

```
print(f"Units Consumed: {units_consumed}")
```

## **OUTPUT:**

The screenshot shows a Visual Studio Code (VS Code) interface. The left sidebar has icons for Explorer, Open Editors, AI Assisted, Outline, and Timeline. The top menu includes File, Edit, Selection, View, Go, Run, Terminal, Help, and AI Assisted. The main area displays a Python script named lab3.py. The code reads previous and current meter readings, determines customer type (domestic or commercial), and calculates bill amount based on consumption units. It handles errors for invalid input and units consumed less than previous units. The bottom status bar shows the file path as C:/Users/sriva/Desktop/3-2 academics/AI Assisted & C:/Users/sriva/appData/local/Programs/Python/Python313/python.exe "C:/Users/sriva/OneDrive/Desktop/lab3.py" and other details like line 31, column 1, and Python 3.13.7.

```
#TASK-1 PYTHON code for reading consumer details and generate a python program it should read previous units and current units and type of customer
# Reading consumer details and calculating units consumed
previous_units = int(input("Enter previous meter reading:"))
current_units = int(input("Enter current meter reading:"))
customer_type = input("Enter customer type (domestic/commercial):").strip().lower()
units_consumed = current_units - previous_units
if units_consumed < 0:
    print("Error: current units cannot be less than previous units.")
else:
    if customer_type == "domestic":
        if units_consumed <= 100:
            bill_amount = units_consumed * 1.5
        elif units_consumed <= 300:
            bill_amount = (100 * 1.5) + (units_consumed - 100) * 2.5
        else:
            bill_amount = (100 * 1.5) + (200 * 2.5) + (units_consumed - 300) * 4.0
    elif customer_type == "commercial":
        if units_consumed <= 100:
            bill_amount = units_consumed * 2.0
        elif units_consumed <= 300:
            bill_amount = (100 * 2.0) + (units_consumed - 100) * 3.5
        else:
            bill_amount = (100 * 2.0) + (200 * 3.5) + (units_consumed - 300) * 5.0
    else:
        print("Error: Invalid customer type.")
        bill_amount = None
    if bill_amount is not None:
        print(f"Units Consumed: {units_consumed}")
```

## **EXPLANATION:**

- This task focuses on collecting correct consumer details required for electricity billing.
  - The program reads previous units, current units, and consumer type from the user.
  - Units consumed are calculated using meter readings, which is the base for billing.
  - This step ensures accurate input handling for further calculations.

## ❖ TASK-2

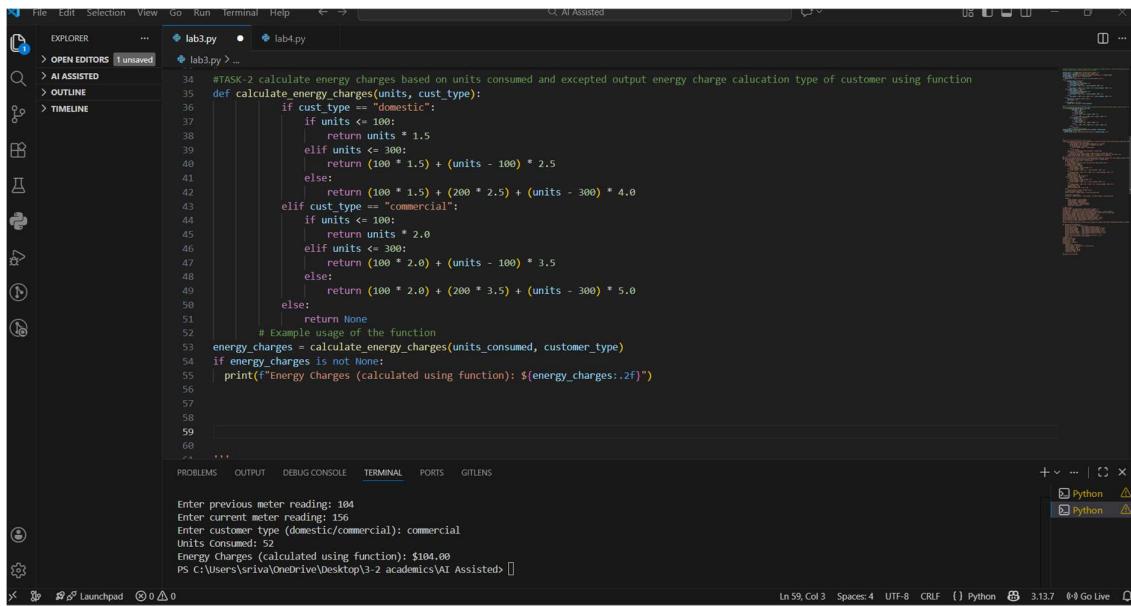
### PROMPT:

calculate energy charges based on units consumed and excepted output energy charge calucation type of customer using function.

### CODE:

```
def calculate_energy_charges(units, cust_type):  
    if cust_type == "domestic":  
        if units <= 100:  
            return units * 1.5  
        elif units <= 300:  
            return (100 * 1.5) + (units - 100) * 2.5  
        else:  
            return (100 * 1.5) + (200 * 2.5) + (units - 300) * 4.0  
    elif cust_type == "commercial":  
        if units <= 100:  
            return units * 2.0  
        elif units <= 300:  
            return (100 * 2.0) + (units - 100) * 3.5  
        else:  
            return (100 * 2.0) + (200 * 3.5) + (units - 300) * 5.0  
    else:  
        return None  
  
# Example usage of the function  
  
energy_charges = calculate_energy_charges(units_consumed, customer_type)  
if energy_charges is not None:  
    print(f"Energy Charges (calculated using function): ${energy_charges:.2f}")
```

## OUTPUT:



```
File Edit Selection View Go Run Terminal Help < - > C:\AI Assisted
OPEN EDITORS 1 unsaved
> AI ASSISTED
> OUTLINE
> TIMELINE
lab3.py
34 #TASK-2 calculate energy charges based on units consumed and excepted output energy charge calucation type of customer using function
35 def calculate_energy_charges(units, cust_type):
36     if cust_type == "domestic":
37         if units <= 100:
38             return units * 1.5
39         elif units <= 300:
40             return (100 * 1.5) + (units - 100) * 2.5
41         else:
42             return (100 * 1.5) + (200 * 2.5) + (units - 300) * 4.0
43     elif cust_type == "commercial":
44         if units <= 100:
45             return units * 2.0
46         elif units <= 300:
47             return (100 * 2.0) + (units - 100) * 3.5
48         else:
49             return (100 * 2.0) + (200 * 3.5) + (units - 300) * 5.0
50     else:
51         return None
52     # Example usage of the function
53 energy_charges = calculate_energy_charges(units_consumed, customer_type)
54 if energy_charges is not None:
55     print(f"Energy Charges (calculated using function): ${energy_charges:.2f}")
56
57
58
59
60
61
62
63
64
65
66
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS GITLENS

Enter previous meter reading: 104  
Enter current meter reading: 156  
Enter customer type (domestic/commercial): commercial  
Units Consumed: 52  
Energy Charges (calculated using function): \$104.00  
PS C:\Users\sriva\OneDrive\Desktop\3-2 academics\AI Assisted

Ln 59, Col 3 Spaces: 4 UTF-8 CRLF {} Python 3.13.7 Go Live

## EXPLANATION:

In this task, energy charges are calculated based on units consumed and consumer type.

Conditional logic is used to apply different tariff rates for domestic, commercial, and industrial users.

## ❖ TASK-3:

### PROMPT:

calculate billing logic must be reusable for multiple consumers and calculate energy charges and fixed charges return calculated values

### CODE:

```
def calculate_bill(units, cust_type):

    energy_charges = calculate_energy_charges(units, cust_type)

    fixed_charges = 50 if cust_type == "domestic" else 100

    if energy_charges is not None:

        return energy_charges + fixed_charges

    else:

        return None
```

```

total_bill = calculate_bill(units_consumed, customer_type)

if total_bill is not None:

    print(f"Total Bill Amount (using reusable function): ${total_bill:.2f}")

print(f"Energy Charges: ${bill_amount - (50 if customer_type == 'domestic' else 100):.2f}")

print(f"Fixed Charges: ${50 if customer_type == 'domestic' else 100:.2f}")

```

### OUTPUT:

The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer:** Shows files `lab3.py` and `lab4.py`.
- Terminal:** Displays the execution of the code and its output.
- Output:** Shows the results of the program execution.

```

#TASK-3 calculate billing logic must be reusable for multiple consumers and calculate energy charges and fixed charges return calculated values
def calculate_bill(units, cust_type):
    energy_charges = calculate_energy_charges(units, cust_type)
    fixed_charges = 50 if cust_type == "domestic" else 100
    if energy_charges is not None:
        return energy_charges + fixed_charges
    else:
        return None

total_bill = calculate_bill(units_consumed, customer_type)
if total_bill is not None:
    print(f"Total Bill Amount (using reusable function): ${total_bill:.2f}")
print(f"Energy Charges: ${bill_amount - (50 if customer_type == 'domestic' else 100):.2f}")
print(f"Fixed Charges: ${50 if customer_type == 'domestic' else 100:.2f}")

```

**Terminal Output:**

```

Enter current meter reading: 123
Enter customer type (domestic/commercial): DOMESTIC
Units Consumed: 10
Energy Charges calculated using function: $28.50
Total Bill Amount (using reusable function): $78.50
Energy Charges: $28.50
Fixed Charges: $50.00
PS C:\Users\sriva\OneDrive\Desktop\3-2 academics\AI Assisted>

```

### EXPLANATION:

Introduces modular programming using user-defined methods.

Separate methods are used to calculate energy charges and fixed charges.

This makes the program reusable and easier to maintain.

Modular design improves code structure and readability.

## ❖ TASK-4:

### PROMPT:

Generate electricity bill including multiple additional charges like fixed charges, customer charges, percentage of electricity duty, duty calculation by improving accuracy

**CODE:**

```
def calculate_electricity_bill(previous_units, current_units, customer_type):  
    units_consumed = current_units - previous_units  
  
    if units_consumed < 0:  
        raise ValueError("Current units cannot be less than previous units.")  
  
    if customer_type == "domestic":  
        if units_consumed <= 100:  
            energy_charges = units_consumed * 1.5  
  
        elif units_consumed <= 300:  
            energy_charges = (100 * 1.5) + (units_consumed - 100) * 2.5  
  
        else:  
            energy_charges = (100 * 1.5) + (200 * 2.5) + (units_consumed - 300) * 4.0  
  
        fixed_charges = 50  
  
        customer_charges = 20  
  
        electricity_duty_rate = 0.05  
  
    elif customer_type == "commercial":  
        if units_consumed <= 100:  
            energy_charges = units_consumed * 2.0  
  
        elif units_consumed <= 300:  
            energy_charges = (100 * 2.0) + (units_consumed - 100) * 3.5  
  
        else:  
            energy_charges = (100 * 2.0) + (200 * 3.5) + (units_consumed - 300) * 5.0  
  
        fixed_charges = 100  
  
        customer_charges = 50  
  
        electricity_duty_rate = 0.10  
  
    else:  
        raise ValueError("Customer type must be domestic or commercial.")  
  
    electricity_duty = energy_charges * electricity_duty_rate
```

```

total_bill = energy_charges + fixed_charges + customer_charges + electricity_duty

return units_consumed, energy_charges, fixed_charges, customer_charges,
electricity_duty, total_bill

def get_int(msg):

    while True:

        value = input(msg)

        if value.strip() == "":
            print("Input cannot be empty. Enter a number.")

        elif not value.isdigit():

            print("Enter only digits.")

        else:

            return int(value)

try:

    previous_units = get_int("Enter previous meter reading: ")

    current_units = get_int("Enter current meter reading: ")

    customer_type = input("Enter customer type (domestic/commercial): ").strip().lower()

    u, e, f, c, d, t = calculate_electricity_bill(previous_units, current_units, customer_type)

    print("\n--- Electricity Bill ---")

    print("Units Consumed:", u)

    print("Energy Charges:", round(e, 2))

    print("Fixed Charges:", f)

    print("Customer Charges:", c)

    print("Electricity Duty:", round(d, 2))

    print("Total Bill Amount:", round(t, 2))

except ValueError as err:

    print("Error:", err)

```

## OUTPUT:

The screenshot shows the VS Code interface with the terminal tab active. The terminal window displays the following output:

```
Enter previous meter reading: 123
Enter current meter reading: 159
Enter customer type (domestic/commercial): domestic
--- Electricity Bill ---
Units Consumed: 36
Energy Charges: 54.0
Fixed Charges: 50
```

The screenshot shows the VS Code interface with the terminal tab active. The terminal window displays the following output:

```
Enter previous meter reading: 
Enter current meter reading: 
Enter customer type (domestic/commercial): 
--- Electricity Bill ---
Units Consumed: 36
Energy Charges: 54.0
Fixed Charges: 50
Customer Charges: 20
Electricity Duty: 2.7
Total Bill Amount: 126.7
```

## EXPLANATION:

This task extends billing by adding additional charges like fixed charges, customer charges, and electricity duty.

Electricity duty is calculated as a percentage of energy charges.

Printing individual charges helps verify calculation accuracy.

This step makes the bill more realistic and detailed.

## ❖ TASK-5:

### PROMPT:

Generate final bill of electricity including all charges with proper formatting and display in python .

### CODE:

```
def display_bill(bill_details):

    print("\n----- Electricity Bill -----")

    print(f"Consumer ID : {bill_details['Consumer ID']}")

    print(f"Units Consumed : {bill_details['Units Consumed']} units")

    print(f"Energy Charges : ₹{bill_details['Energy Charges']:.2f}")

    print(f"Fixed Charges : ₹{bill_details['Fixed Charges']:.2f}")

    print(f"Customer Charges : ₹{bill_details['Customer Charges']:.2f}")

    print(f"Electricity Duty : ₹{bill_details['Electricity Duty']:.2f}")

    print("-----")

    print(f"Total Bill Amount : ₹{bill_details['Total Bill']:.2f}")

    print("-----\n")

# Example usage

consumer_id = "C12345"

previous_units = 500

current_units = 750

consumer_type = "domestic"

bill_details = {

    "Consumer ID": consumer_id,

    "Units Consumed": current_units - previous_units,

    "Energy Charges": 625.00,

    "Fixed Charges": 50.00,

    "Customer Charges": 20.00,

    "Electricity Duty": 31.25,
```

```

    "Total Bill": 726.25

}

display_bill(bill_details)

```

### OUTPUT:

The screenshot shows a code editor interface with two files open: lab3.py and lab4.py. The lab3.py file contains a function definition for 'display\_bill' which prints various bill details. The lab4.py file contains an example usage of this function with specific consumer data.

```

lab3.py
130 def display_bill(bill_details):
131     print("----- Electricity Bill -----")
132     print(f"Consumer ID : {bill_details['Consumer ID']}")
133     print(f"Units Consumed : {bill_details['Units consumed']} units")
134     print(f"Energy Charges : ₹{bill_details['Energy charges']:.2f}")
135     print(f"Fixed Charges : ₹{bill_details['Fixed charges']:.2f}")
136     print(f"Customer Charges : ₹{bill_details['Customer charges']:.2f}")
137     print(f"Electricity Duty : ₹{bill_details['Electricity duty']:.2f}")
138     print("-----")
139     print(f"Total Bill Amount : ₹{bill_details['Total Bill']:.2f}")
140     print("-----\n")
141
142 # Example usage
143 consumer_id = "C12345"
144 previous_units = 500
145 current_units = 750
146 consumer_type = "domestic"
147 bill_details = {
148     "Consumer ID": consumer_id,
149     "Units consumed": current_units - previous_units,
150     "Energy charges": 625.00,
151     "Fixed charges": 50.00,
152     "Customer charges": 20.00,
153     "Electricity Duty": 31.25,
154     "Total Bill": 726.25
155 }
156 display_bill(bill_details)
157

```

The output window shows the printed bill details:

```

Consumer ID : C12345
Units Consumed : 250 units
Energy Charges : ₹625.00
Fixed Charges : ₹50.00
Customer Charges : ₹20.00
Electricity Duty : ₹31.25
-----
Total Bill Amount : ₹726.25

```

### EXPLANATION:

final electricity bill by combining all charge components.

The total bill amount is calculated by adding EC, FC, CC, and ED.

The output is displayed in a neat, bill-like format for clarity.

This task demonstrates a complete, real-world electricity billing application